VIEWBEAP
Boundary Element Analysis Program
Post-processor

Technical Reference

Cube Systems Inc.

MICROMEDIA

Revised
July 11, 1991

TABLE OF CONTENTS

1. INTRODUCTION 3
    1.1 DESCRIPTION OF THE SYSTEM 3
    1.2 DESIGN PLATFORM 3
2. GRAPHICS MODULE 5
    2.1 FUNCTION SPECIFICATIONS 5
    2.2 DESIGN STRUCTURE 6
    2.3 SPEED CONSIDERATIONS 7
    2.4 PRIMITIVE DEFINITIONS 8
    2.5 STEREOSCOPIC VISION DRIVERS 12
    2.6 HIDDEN FACE REMOVAL TECHNIQUES 13
    2.7 SHADING TECHNIQUES USED 18
ANIMATE.C 22
    void doauxiliary(HANDLE hWnd) 22
    long FAR PASCAL auxDlgProc() 22
    long FAR PASCAL auxilliaryproc() 22
    BOOL freadmy(FILE *fp, LPSTR lpointer, long count) 22
    BOOL fwritemy(FILE *fp, LPSTR lp, long count) 22
    DWORD PASCAL lread (HANDLE fh, VOID FAR *pv, DWORD ul) 22
    DWORD PASCAL lwrite (HANDLE fh,VOID FAR *pv, DWORD ul) 23
    void animateLogAddFrame(DISPLIST *dispList) 23
    void animateLogSetDispList(DISPLIST *dispList, int frame) 23
    HANDLE animateLogStart(DISPLIST *dispList) 23
    int animateLogEnd(DISPLIST *dispList) 23
    void animateprocess(DISPLIST *dispList) 23
    LPSTR anigetfilename(int which) 23
    void animateWriteBits(HANDLE fp, LPSTR ptr, long size) 24
    void animateReadBits(HANDLE fp, LPSTR ptr, long bitsize) 24
    HANDLE ReadDibBitmapInfo (HANDLE fh) 24
    BOOL DibInfo (HANDLE hbi,LPBITMAPINFOHEADER lpbi) 24
    HANDLE OpenDIB (LPSTR szFile) 24
    HANDLE DibFromBitmap (HBITMAP hbm) 24
    BOOL WriteDIBits (LPSTR szFile, HANDLE hdib) 24
    void savebits(HBITMAP bm, int which) 25
    void animatePaintBuffer(DISPLIST *dispList, ZDisplay *disp, int which) 25
    void aniScreenToFile(DISPLIST *dispList) 25
    void animatePlayBack(DISPLIST *dispList) 25
    BOOL FAR PASCAL aniOptions() 25
    int aniupdatepicture(HANDLE hwnd, int which) 25
    BOOL FAR PASCAL Animate() 25
    long FAR PASCAL animateproc() 26
    voiddoanimate(HANDLE hwnd) 26
    void animaterepeat() 26
BEAPINIT.C 27
    BOOL initWindowClass(hInstance) 27
    int registerUserWind() 27
BEAPMAIN.C 28

92pp

92-01268/

long FAR PASCAL BeapWndProc( ) 28
void destroyAll() 28
void mainGenImage() 28
long FAR PASCAL AxisWndProc() 28
BPCOLOR.C 29
short setupcolor() 29
BPFILE.C 30
void mainOverlay(DISPLIST *dispList,HDC image) 30
long FAR PASCAL CommandHandle() 30
BOOL FAR PASCAL About() 30
HANDLE FAR PASCAL LoadDlg() 30
HANDLE FAR PASCAL SaveDlg() 30
BPRESULT.C 31
ResultPlane *resultGetPlane(int which) 31
int resultEnumPlanes() 31
void resultDelPlane(int which) 31
int resultAddPlane(ResultPlane *plane) 31
char *resultFName() 31
void resultLoadPlanes() 31
int resultSavePlanes() 31
COPY.C 32
BOOL MEM_CMP(char *m1, char *m2, long size) 32
void MEM_CPY(char *dst,char *src,unsigned int size) 32
void FAR *MEM_ALLOC(long size) 32
void FAR * MEM_FREE(void FAR *pointer) 32
void FAR *MEM_REALLOC(void FAR *pointer, long size) 32
BOOL bitmaptoclip(HWND hWnd, int X,int Y,int X2, int Y2) 32
BOOL cliptoscreen(HWND hWnd,int X,int Y, int X2, int Y2) 33
DATA.C 34
ELEMCODE *dataGetEcode(unsigned int n) 34
EXTRADATA *dataGetExtra() 34
NODEDATA *dataGetNode(unsigned int n) 34
ELEMDATA *dataGetElem(unsigned int n) 34
unsigned int dataGetNumNodes() 34
unsigned int dataGetNumElems() 34
void dataDelExtra() 34
void dataClear() 35
void dataAddExtra(float xOff, float yOff, float zOff) 35
unsigned int dataAddNode(unsigned int n, float x, float y, float z) 35
unsigned int dataAddElem(unsigned int n, short sides, unsigned int *nodes, short color, char *description) 35
void dataCalcNodeVisibility(DISPLIST *dispList) 35
DIALOGMY.C 36
BOOL GetDlgItemFloat(HWND hDlg,short IDMnumber,float *number) 36
void SetDlgItemFloat(HWND hDlg, short IDMnumber, float number) 36
BOOL GetDlgItemInteger(HWND hDlg,short IDMnumber,int *number) 36
DISP.C 37
void dispRecalc() 37
void dispRepaint() 37

void dispListMoved() 37
long dispListSetOverlay() 37
long dispListSetMoveProc() 37
void dispListSetAngles(DISPLIST *dispList, float angXY, float angYZ) 37
void dispListGetAngles(DISPLIST *dispList, float *angXY, float *angYZ) 37
void dispListSetOrigin(DISPLIST *dispList, float x, float y, float z) 38
void dispListSetViewSize(DISPLIST *dispList, long viewSize) 38
void dispListSetShading(dispList, shadedModel) 38
HANDLE dispListCreate(HWND parent,HWND view, void (*recalcProc)()) 38
HANDLE dispListDestroy(HANDLE hdispList) 38
void dispDraw(DISPLIST *dispList, HDC image, int width, int height) 38
void dispDrawBuffered(DISPLIST *dispList,HDC image, int width, int height) 38
void dispPrintBuffered (DISPLIST *dispList,HDC image, int width, int height, int scale) 39
int dispCheckResultPlanes(int id, ResultPlane *plane, struct extremesTag *extremes) 39
void dispGenImage() 39
void dispGenAxisOverlay(DISPLIST *dispList,HDC image) 39
void dispGenAxis(DISPLIST *dispList) 39
void dispPaint(DISPLIST *dispList, HDC image, int width, int height) 39
void dispPrint(DISPLIST *dispList, HDC image, int width, int height, int scale) 39
void dispForceRecalc(DISPLIST *dispList) 40
void dispUpdate(DISPLIST *dispList) 40
DRAW.C 41
void xv_text(HDC hdc,int x, int y, int color, int op,HFONT font,LPSTR str) 41
void xv_polygon_2(HDC hdc,int *numPts,pr_pos *vlist,int color,int op) 41
void xv_vector(HDC hDC,int x1,int y1,int x2,int y2,int op,int pixel) 41
ELEMST.C 42
float elemStressGetStress(int which,StressTypes type, int x, int y) 42
void elemstfree() 42
BOOL elemStressLoad(int miningStep) 42
void domeshsize(HWND hwnd,DISPLIST *dispList) 42
long FAR PASCAL meshsizeproc() 42
void elemDispGenMeshCoeff(float (*coeff)[9], int ix, int iy, float f) 42
void elemDispMult(float *U, float N[9], float (*Ui)[ ], int n) 43
void elemDispGenElemMesh(int elemNum, float (*nodes)[4], int calcStress, float factor, int calcDisp, int stressType, float (*coeff)[9], int ix, int iy) 43
EYEWND.C 44
BOOL initeye() 44
void updateeyepalet() 44
void updateeyeaction() 44
void draweye(HANDLE hwnd,int x, int y) 44
void drawex(HANDLE hwnd,int x, int y) 44
void reseteye() 44
void eyeset(DISPLIST *dispList,BOOL flag) 44
void eyesetang(float x, float y) 45
void eyeresetang() 45
void correctrect(HANDLE hwnd) 45
void moveeyeorex(HANDLE hwnd,BOOL flag) 45

void FAR PASCAL getresultrect() 45
void doeyeok(DISPLIST *dispList) 45
void eyecommandhandle() 45
long FAR PASCAL EyeWndProc() 46
long FAR PASCAL EyePaletWndProc() 46
long FAR PASCAL EyeXYWndProc() 46
long FAR PASCAL EyeXZWndProc() 46
long FAR PASCAL EyeYZWndProc() 46
long FAR PASCAL EyeISOWndProc() 46
GETPOINT.C 47
void drawkrzyz(HANDLE hwnd,int x, int y) 47
void drawkrzyzall() 47
void initgetpoint(HANDLE hdlg,float x,float y,float z) 47
int getPointState() 47
int getPointStress(float fx, float fy, float fz) 47
void getPointCalcStress(HANDLE hdlg) 47
void getPointSetLists(HANDLE hdlg,float *value, int count) 47
BOOL FAR PASCAL getpointcubedlg() 48
void getpointsetcube(HANDLE hdlg) 48
BOOL FAR PASCAL getpointm507dlg() 48
void getpointsetm507(HANDLE hdlg) 48
BOOL FAR PASCAL getpointdlg() 48
void getpoint(HANDLE hwnd) 48
GETST.C 49
static int _divideGetRealCoors() 49
static int _findCell(FindValInfo *info) 49
static int _getCell(FindValInfo *info, int which) 49
float getStressValue(FindValInfo *rootInfo, float fx, float fy, float fz, float *distance) 49
char *getStressFName(int which, int miningStep) 49
FindValInfo *getStressLoadCube(FindValInfo *info, int regionNumber) 49
FindValInfo *getStressLocateCube(FindValInfo *rootInfo, float fx, float fy, float fz) 49
FindValInfo *getStressStart(StressTypes type, int miningStep) 50
void getStressEnd(FindValInfo *info) 50
CalcCube *getStressCalcCube(int which) 50
int getStressNumCalcCubes() 50
BOOL getStressEnumCalcCubes(int (*func) (), char *data) 50
void getStressDelCalcCube(int which) 50
int getStressAddCalcCube() 50
void getStressUpdateCalcCube(int which, CalcCube *data) 51
void getStressLoadCalcCubes() 51
void getStressSaveCalcCubes() 51
void findStressMaxMin() 51
GRAPHWND.C 52
void FAR PASCAL dogeometria() 52
long FAR PASCAL paintgraphWndProc() 52
void drawbox(HWND hWnd,BPRECT rect,int x,int y,BOOL draw) 52
void drawline(HWND hWnd,BPRECT rect,int x,int y,BOOL draw) 52

void dozooming(HWND hWnd,DISPLIST *dispList,InputBox result) 52
void dorotating(HWND hWnd,DISPLIST *dispList,BPRECT rect) 53
void dopaning(HWND hWnd,DISPLIST *dispList, BPRECT rect) 53
void dokeything() 53
HIDEPL.C 54
BOOL inithideplane() 54
void updatehideplanepalet() 54
void updatehideplaneaction() 54
BOOL selectResultSetVisibility(DISPLIST *dispList, DISPPLATE *plate) 54
int selectResultDrawPlane() 54
void hideplanewndcommand() 55
long FAR PASCAL HidePlaneWndProc() 55
long FAR PASCAL hideplanepWndProc() 55
long FAR PASCAL painthideplaneWndProc() 55
long FAR PASCAL axishideplaneWndProc() 55
HIDEWND.C 56
BOOL inithide() 56
void updatehidepalet() 56
void updatehideaction() 56
void hidePlateSetColor(DISPLIST *dispList, DISPPLATE *plate) 56
void hidePlateState(HANDLE hwnd, DISPLIST *dispList, InputBox results) 56
void hidePlateUpdateData(DISPLIST *dispList, DISPPLATE *plate, char *data) 56
void FAR PASCAL dosethide() 56
void hidewndcommand() 57
long FAR PASCAL HideWndProc() 57
long FAR PASCAL hidepWndProc() 57
long FAR PASCAL painthideWndProc() 57
long FAR PASCAL axishideWndProc() 57
LEGEND.C 58
void legendChangeType(HWND hDlg, WORD wParam) 58
void legendchangefont(HWND hDlg, WORD wParam) 58
void legendchangelocation(HWND hDlg, WORD wParam) 58
void legendchangetextlocation(HWND hDlg, WORD wParam) 58
BOOL legendokey(HANDLE hdlg) 58
void legendGetOld(HANDLE hwnd) 59
short legendCheckChange(HANDLE hwnd) 59
BOOL FAR PASCAL legendDlgProc() 59
short dolegend(HWND hwnd) 59
HFONT legendGetFont(HDC image) 59
void legendTextOverlay(DISPLIST *dispList, HDC image) 59
void legendOverlay(DISPLIST *dispList,HDC image) 59
int legendGetStressColor(float value, StressTypes type) 59
OPENFILE.C 60
int loadFile() 60
void FAR PASCAL dispError(HWND hwnd, LPSTR buf) 60
int saveFile() 60
int saveAll(char *dir) 60
int loadAll(dir) 60
PAINTWND.C 61

BOOL initpaint() 61
void updatepalet() 61
void updateaction(int id) 61
BOOL setColorOfPlate(DISPLIST *dispList, DISPPLATE *plate) 61
void setpaintcolor(HANDLE hwnd,DISPLIST *dispList,InputBox results,int colornr) 61
void setColorUpdateData(DISPLIST *dispList, DISPPLATE *plate, char *data) 61
long FAR PASCAL paintp1WndProc() 62
long FAR PASCAL paintp2WndProc() 62
long FAR PASCAL axispaintWndProc() 62
long FAR PASCAL paintpaintWndProc() 62
PALET.C 63
void getAction(int x, int y, PALLET NEAR *pallet) 63
void palletup(PALLET NEAR *pallet,int *x,int *y) 63
void palletdn(PALLET NEAR *pallet,int *x,int *y) 63
PALLET NEAR *paletinit(HWND hwnd,int offx, int offy, int width, int height,int wn, int hn, int oldaction, int action) 63
void paletupdate(PALLET NEAR *palet, HWND hwnd, short *tableup) 63
void paletupdateaction(HWND hwnd, PALLET NEAR *palet, short *Tableup, short *Tabledn) 64
BOOL paletmouse() 64
PERSWND.C 65
BOOL initpers() 65
void updateperspalet() 65
void updatepersaction(BOOL flag) 65
long FAR PASCAL PersWndProc() 65
PLANEWND.C 66
BOOL initplane() 66
void updateplanepalet() 66
void updateplaneaction() 66
int checkplane(HANDLE hwnd) 66
void defResultUpdateChildren() 66
BOOL defResultDelPlane(DISPLIST *dispList, DISPPLATE *plate) 66
BOOL defResultDelState() 67
BOOL defResultMoveStartResult(DISPLIST *dispList, DISPPLATE *plate) 67
void defResultOverPaintPlane(HANDLE view, ResultPlane *plane) 67
void defResultOverPaintPlane(ResultPlane *plane) 67
int defResultMoveState() 67
void doresultsize() 67
int defResultSizeState() 67
void defResultSetPlaneLoc() 67
void defResultSetPlaneSize() 68
void defResultSetPanelData(HANDLE hdlg,ResultPlane *plane) 68
BOOL planegetvalue(HANDLE hdlg,ResultPlane *plane) 68
BOOL FAR PASCAL editresultDlg() 68
void defResultEditPlaneData(int planeId) 68
BOOL defResultEditPlane(DISPLIST *dispList,DISPPLATE *plate) 68
void defResultAddPlane() 68
void defResultDrawPlane(int id,ResultPlane *plane, DISPLIST *dispList) 68

void defResultDrawCube(int id, CalcCube *cube, DISPLIST *dispList) 69
void defResultGenImage(DISPLIST *dispList) 69
void resetplane() 69
void planesetpananglesview(HANDLE hwnd,DISPLIST *dispList) 69
void planesavepanview(HANDLE hwnd,DISPLIST *dispList) 69
int defResultEditState() 69
BOOL FAR PASCAL maxminDlgProc() 69
int doshowmaxmin(HANDLE hWnd,DISPLIST *dispList) 69
void planecommandhandle() 70
long FAR PASCAL PlaneWndProc() 70
long FAR PASCAL PlanePaletWndProc() 70
void doplanegeo(HWND hwnd,unsigned message,WORD wParam,LONG lParam) 70
long FAR PASCAL PlanXYWndProc() 70
long FAR PASCAL PlanXZWndProc() 70
long FAR PASCAL PlanYZWndProc() 70
long FAR PASCAL PlanISOWndProc() 70
PRINT.C 71
void printInitFirst() 71
void printChangePrinter(HANDLE hwnd) 71
BOOL FAR PASCAL changePrintDlg() 71
HDC getPrinterDC() 71
BOOL FAR PASCAL GetInitializationData(HWND hwnd, short which) 71
BOOL FAR PASCAL AbortDlg() 71
int FAR PASCAL AbortProc(HDC hPr, int Code) 71
void fontCorrect(HDC hdc) 72
void fontSetBack() 72
short printOptions(DISPLIST *dispList) 72
BOOL FAR PASCAL PrintDlgProc (hwnd, message, wParam, lParam) 72
void doprint(DISPLIST *dispList) 72
RESWND.C 73
BOOL initresult() 73
void updateresultpalet() 73
void updateresultaction() 73
void defRegionUpdateChildren() 73
BOOL defRegionDelRegion(DISPLIST *dispList, DISPPLATE *plate) 73
BOOL defRegionDelState() 73
BOOL defRegionMoveStartRegion(DISPLIST *dispList, DISPPLATE *plate) 73
void defRegionOverPaintCube(HANDLE hwnd, CalcCube *cube) 74
void defRegionOverPaintCube(CalcCube *cube) 74
void defRegionMoveState() 74
void doregionsize(HANDLE hwnd, DISPLIST *dispList, int *flag) 74
int defRegionSizeState() 74
void cubeeditinit(HWND hdlg,CalcCube *cube) 74
BOOL cubegetvalue(HWND hdlg,CalcCube *cube) 74
BOOL FAR PASCAL editregionDlg() 74
void defRegionEditRegionData(int cubeId) 75
BOOL defRegionEditRegion(DISPLIST *dispList,DISPPLATE *plate) 75
void defRegionAddRegion() 75
BOOL defRegionDrawCube(int id, CalcCube *cube, DISPLIST *dispList) 75

void defRegionGenImage(DISPLIST *dispList) 75
void resetcube() 75
void cubesetpananglesview(HANDLE hwnd,DISPLIST *dispList) 75
void cubesavepanview(HANDLE hwnd,DISPLIST *dispList) 76
int defregioneditstate() 76
void cubecommandhandle() 76
long FAR PASCAL ResultWndProc() 76
long FAR PASCAL ResultPaletWndProc() 76
long FAR PASCAL ReXYWndProc() 76
long FAR PASCAL ReXZWndProc() 76
long FAR PASCAL ReYZWndProc() 76
long FAR PASCAL ReISOWndProc() 77
SELECT.C 78
int selectClearObj(DISPLIST *dispList, DISPGENOBJ *obj, char *data) 78
void selectClearAll(DISPLIST *dispList, int type) 78
BOOL selectIsInRect(BPRECT *rect, int x, int y) 78
int selectRectCheckObj(DISPLIST *dispList, DISPGENOBJ *obj, BPRECT *rect) 78
void selectInRect(DISPLIST *dispList, int type, BPRECT *rect) 78
int selectPointCheckObj(DISPLIST *dispList, DISPGENOBJ *obj, SelectPoint *pt) 78
void selectAtPoint(DISPLIST *dispList, int type, int x, int y) 78
int selectEnumFunct(DISPLIST *dispList, DISPGENOBJ *obj, SelectEnumData *data) 79
void selectEnum(DISPLIST *dispList,DispPrimitiveType type,void (*funct) ()) 79
void selectEnumFirst(DISPLIST *dispList,DispPrimitiveType type,void (*funct) ()) 79
SOLID.C 80
BOOL initsolid() 80
void updatesolidaction(HANDLE hwnd) 80
long FAR PASCAL SolidDlgProc() 80
long FAR PASCAL solidwndproc() 80
SPEED.C 81
BOOL initspeed() 81
void updatespeedpalet(HANDLE hwnd) 81
void updatespeedaction(HANDLE hwnd) 81
long FAR PASCAL speedDlgProc() 81
long FAR PASCAL speedproc() 81
STEP.C 82
BOOL initstep() 82
void updatesteppalet(HANDLE hwnd) 82
void updatestepaction(HANDLE hwnd) 82
long FAR PASCAL StepDlgProc() 82
long FAR PASCAL stepwndproc() 82
STRESST.C 83
BOOL initstresstype() 83
void updatestresstypepalet(HANDLE hwnd) 83
void updatestresstypeaction(HANDLE hwnd) 83
void setlaststresstype(DISPLIST *dispList) 83

int getlaststresstype() 83
long FAR PASCAL StresstypeDlgProc() 83
long FAR PASCAL stresstypeproc() 83
STRESSV.C 84
BOOL initstressview() 84
void updatestressviewpalet(HANDLE hwnd) 84
void updatestressviewaction(HANDLE hwnd) 84
long FAR PASCAL StressviewDlgProc() 84
long FAR PASCAL stressviewproc() 84
THREED.C 85
BOOL FAR PASCAL threeDDlg() 85
int dodistance(HANDLE hwnd, DISPLIST *dispList, DISPLIST *axisdispList) 85

1. INTRODUCTION

1.1 DESCRIPTION OF THE SYSTEM

ViewBeap Post-Processor was developed by Cube Systems Consulting Services Inc. of Ottawa, represents the state-of-the-art in graphics software design in present day computer systems. The system was generated to complement BEAP a boundary element analysis program used to determine stresses strains and displacements around mining stopes in a mine site defined in a sub-soil region. The graphics system complements the program, in giving mining engineers visual representation of critical regions and the validity of the computer results, usually a very difficult and a time consuming task using direct numerical output.

The system was developed in three stages. Each stage defined by modules are interconnected to each other giving the user various permutations and control over how the graphical representation will be displayed on the screen. These stages, listed below, will be explained in detail in the following sections.

1- GRAPHICS MODULE

The module where the graphics functions are implemented.

2- SYSTEM MODULE

The module where overall windowing and function calls are implemented in an object oriented fashion.

3- TECHNICAL MODULE

The module where engineering stress strain and displacement representations, subdivisioning techniques are implemented.

1.2 DESIGN PLATFORM

The program was developed using a 386 Based personal computer Sparcstation with 8 megabytes of RAM and 100 megabytes of Hard disk storage with a Enhanced VGA graphics subsystem using a 14" colour monitor.

The system was simultaneously developed in two environments. Microsoft Windows and XView which is based on X-Windows X11 standard. Although in certain functions XView proved to be faster, the overall versatility of the X-Windows system was favoured heavily due to the requirement of portability to i386 and i486 platforms.

The program was entirely written in C and the coding was designed an grouped in such a way that easy porting can be possible to up and coming environments.

2. GRAPHICS MODULE

This section describes the basic principles used for the design of the graphics system for BEAP post processor. We assume that the reader has some prior background in computer graphics and familiar with fundamental geometrical concepts and methods.

2.1 FUNCTION SPECIFICATIONS

The graphics functions implemented can be summarized as follows:

1- Rotations
2- Panning and moving
3- Partial hide / show capabilities
4- Full hidden line removal
5- Full hidden object or plane removal
6- Isometric and perspective representations
7- Eye orientation and direction changes.
8- Real time graphics actions
9- Full record and play back animation.
10- Grouped colouring of faces
11- Shading
12- Stereoscopic display

The graphics system was designed to represent the following display models:

SLOW MODE : No hardware dependent functions was used. Primary technique used is front to back Z-buffering. This technique requires less number of pixels to be painted and therefore is faster then other methods. Although the technique requires a lot of system memory it does not depend on floating point math for the calculation of

the intersection of polygons. Implemented functions using this mode are listed below:

Wire Frame
Filled Surfaces
Shaded Filled surfaces
Stereoscopic Wire Frame
Stereoscopic Filled
Stereoscopic Shaded Filled
Gouraud Shading
Stress Contouring

FAST MODE : Some X Graphic functions were used. Back to front generation depth sorting method is used instead of Z-Buffering technique. Implemented functions using this mode are listed below:

Wire Frame
Filled Surfaces
Shaded Filled Surfaces
Stereoscopic Wire Frame
Stereoscopic Filled
Stereoscopic Shaded Filled

2.2 DESIGN STRUCTURE

The graphics system is designed based on definitions of a display lists. The structure of a display list describes the graphics operation to be performed on a specific object. Please note that the graphics objects are always treated as 3-D geometrical representations in the BEAP Post processor. A general description of a display list is given below.

Display List

- *Attributes*
  - *Screen orientation*
  - *Objects within display*
    - *Lines*
    - *Polygons*
    - *Text*
- *Parameters*
  - *Spatial orientation*
  - *Eye position*
  - *Eye destination*
  - *Shading Model*
  - *3-D Stereoscopic effect*
  - *Isometry / Perspective modes*
  - *Current rotations*

2.3 SPEED CONSIDERATIONS

The graphics system is designed to be as responsive as possible in order to give users the capability to manipulate the graphics real-time. The speed of certain graphics functions do require some time lag to be fully displayed. Gouraud shading, stress contouring and some 3-D stereoscopic shading models require a lot of graphical computations and therefore takes longer to be generated as compared to other graphics models used in the program. Although it was possible to generate graphics for these models almost instantaneously using the graphics hardware available, this was not selected as the design route. The main reasoning behind this decision is to preserve compatibility with systems and to eliminate hardware dependency.

In order to increase speed of graphics based on mathematical vector computations, no floating point arithmetic was used. All computations are based on integer arithmetic. The program was written so that it also eliminated intensive multiply and divide operations by using integer arithmetic. On such operation is illustrated below.

multiplication of two numbers a and b is required ;

a x b

define a new number b'

b' = b x 2048

use b' to operate

a x b'

to find the true number do not use

(a x b') / 2048

but perform bit operations

(a x b') > > 11

which means result shifted down 11 bits

2.4 PRIMITIVE DEFINITIONS

2.4.1 LINES

DDA algorithm (digital differential analyzer) was implemented initially as a trial basis. DDA is an algorithm for calculating pixel positions along a line. This is accomplished by taking unit steps with one coordinate and calculating corresponding values for the other coordinate. The algorithm, is better than directly implementing the line equation and eliminates the multiplication in the line equation by taking the advantage of raster characteristics, so that unit steps are taken in either the x or y directions to the next pixel location along the line. However, the calculations have to use floating point arithmetic and rounding operations which slows the system. The algorithm was therefore removed and replaced by a derivative of Bresenham's line algorithm.

Bresenham's line algorithm uses incremental moves and integer mathematics. It is accepted that this method is more efficient in determining pixel positions by finding the closest integer coordinates of the actual line path.

The method can be summarized as follows:

Input line endpoints and store left point (x_1,y_1) along with right point (x_2,y_2)

Select left point as the first point to be displayed.

Calculate

$\Delta_x = x_2 - x_1$

and

$\Delta_y = y_2 - y_1$

Define P_1

$P_1 = 2\Delta_y - \Delta_x$

If $P_1 < 0$ then

the next point to be set is (x_{i+1}, y_1)

else

the next point is (x_{i+1}, y_{i+1})

Repeat

Continue to increment the x coordinate by unit steps.

if $P_i < 0$ At position x_{i+1} the coordinate to be selected is y_i else y_{i+1}

Parameter is also varied

if $P_i < 0$ then the next parameter is $P_{i+1} = P_i + 2\Delta_y$

else the parameter is $P_i + 1 = P_i + 2(\Delta_y - \Delta_x)$

If $P_{i+1} < 0$ the next y coordinate is y_{i+1} else it is $y_{i+1} + 1$

until $x = x_2$

Better results were obtained for speed by modifying Bresenham's line algorithm. This was done by introducing a buffer value B_x into the equation where B_x is defined as $B_x = \Delta_y / 2$ and the parameter P_i was replaced by B_x and the overall formulation was modified. BEAP postprocessor uses the modified Bresenham's algorithm. The procedure used is given below:

```
bx = dy/2
x = x1
for each raster line
        if (( bx + = dx) > = dy) {
                x + +
                bx -= dy
        }
```

2.4.2 POLYGONS

The polygon drawing primitives were developed by showing extensive care not to duplicate and draw over existing pixels. In order to achieve this a modified version of Scan line algorithm was used.

Scan-line algorithm uses the intersections between area boundaries and scan lines to identify pixels that are inside the area. The figure as shown below illustrate an area outline and an individual scan line passing through a polygon. Pixel positions along the scan line that are within the polygon definition are set to the intensity or color values specified in the application program. Taking each scan line in turn, a scan-conversion algorithm locates the intersection points of the scan line with each edge of the area to be filled. Proceeding from left to right, intersections are paired, and the intervening pixels are set to the specified fill intensity or color. In the illustration the four intersection points with the polygon boundaries define two stretches of interior pixels.

When a scan line intersects a polygon vertex, it may require special handling. A scan line passing through a vertex intersects two polygon edges at the position, adding two points to the list of intersections for the scan line. As shown in the illustration scan line 1 intersects a polygon boundary four times. Two interior stretches are defined: one from the left boundary to the vertex, and a second from the vertex to the right edge of the polygon. But scan line 2 generates five intersections with polygon edges, and resultant pairs do not correspond to the polygon interior.

To fill a polygon correctly, its overall topology must be considered. If the vertices of the polygon are specified in clockwise order, scan line 2 as shown in the illustration intersects a vertex whose connecting edges are monotonically decreasing in the y direction. When successive edges of the polygon are monotonically increasing or decreasing, a correct determination of interior points along a scan line is obtained by recording only one intersection point for the vertex. The vertex intersection on scan line 1 in this figure connects two lines with opposite y directions. One line has decreasing y -coordinate values, and the other has increasing y -coordinate values. When such a local minimum (or a local maximum) is encountered by a scan line, two intersection points should be generated to correctly identify interior pixels along the scan line.

The polygon boundaries were drawn using the modified Breshenham's algorithm. A summary of the procedure is given below:

```
x = x1
bx = dy / 2
for each raster line
if ((bx + = dx ) > = dy ) {
        x+ = bx/dy
        bx = bx mod dy
}
```

The direct fills are done using the same technique. If the shading is not included the direct fill is implemented by switching colours to a solid colour in between the regions where the polygon sides enclose. The resulting image will be filled polygons.

Since it is difficult to differentiate between surface orientation in a three dimensional modelling problem, it was necessary to draw the polygon border using a different colour. In order to achieve this effect, the dual points which intersect the raster lines were switched on using the border colours. This generated the fill and the border at the same time, thus creating the required effect.

Some cases the border around the polygon and the polygon fill compete for the depth value simultaneously. Graphics incompatibilities arise where integer arithmetic results in some accuracy losses. To avoid this discrepancy a z-buffering technique was implemented. To do this, one should find the z coordinate for every point. More description of the technique is illustrated in the following sections.

Every polygon point was defined along with 4 dimensions. Namely these are x y and z coordinates along with the colour definition. Gouraud shading was implemented by varying the fourth dimension.

2.4.3 TEXT PRIMITIVES

System text was used for all purposes in Beap post-processor. Special routines for this purpose was not generated.

2.5 STEREOSCOPIC VISION DRIVERS

Stereoscopic vision technique is a known concept in the motion picture industry. However, implementation of the method in the software applications was not available. For the implementation of the technique for CAD applications Cube Systems holds pending patent application.

The mathematical implementation of the concept is straightforward. It is based on separating the image into two each having one chromatic colour. For BEAP implementation these colours are selected to be Cyan and Red. A pair of filtering glasses filter one of the colours for each eye, thus superimposing the image as perceived by the user. The calculations for the separating of the images on the screen depends on three factors which are described below:

Z = The distance of the object from the users location.
d = The distance of the user from the screen.
w = Distance between the actual vision median and the eye focal point.

Correction factor for each eye thus can be calculated by

$$S_x = w (z - d) / z$$

Using these variables one can calculate the distance between the two images on the screen.

2.6 HIDDEN FACE REMOVAL TECHNIQUES

2.6.1 BACK FACE REMOVAL

A simple object-space method for identifying the **back faces** of objects is based on the equation of a plane:

$$Ax + By + Cz + D = 0$$

Any point (x',y',z') specified in a right-handed coordinate system is on the "inside" of this plane if it satisfies the inequality

$$Ax' + By' + Cz' + D < 0$$

If the point (x',y',z') is the viewing position, then any plane for which inequality is true must be a back face. This is, it is one that we cannot see from the viewing position.

We can perform a simpler back-face test by looking at the normal vector to a plane described by the surface equation. This normal vector has Cartesian components (A, B, C). In a right-handed viewing system with the viewing direction along the negative z- axis, the normal vector has component C parallel to the viewing direction. If C < 0, the normal vector points away from the viewing position, and the plane must be a back face.

By examining parameter C for the different planes defining an object, we can immediately identify all the back faces. For a single convex polyhedron, such as a pyramid, this test identifies all the hidden surfaces on the object, since each surface is either completely visible or completely hidden. For other objects, more tests need to be carried out to determine whether there are additional faces that are totally or partly hidden. Also, we may need to determine whether some objects are partially or completely obscured by other objects. In general, back-face removal can be expected to eliminate about half of the surfaces in a scene from further visibility tests.

2.6.2 Z- BUFFERING

Another method used in the image-space approach to eliminating hidden surfaces is the depth-buffer method, also called the z-buffer method. Basically, this algorithm tests the visibility of surfaces on point at a time. For each pixel position (x, y) on the viewing plane, the surface with the smallest z coordinate at that position is visible. Three surfaces at varying depths with respect to position (x, y) in a left-handed viewing system. Surface S_1 has the smallest z value at this position, so its intensity value at (x, y) is saved.

Two buffer areas are required for implementation of this method. A depth buffer is used to store z values for each (x, y) position as surfaces are compared, and the refresh buffer stores the intensity values for each position.

This method can be implemented conveniently in normalized coordinates, with depth values varying from 0 to 1. Assuming that a projection volume has been mapped into a normalized parallelepiped view volume, the mapping of each surface onto the view plane is an orthographic projection. The depth at points over the surface of a polygon is calculated from the plane equation. Initially, all positions in the depth buffer are set to 1 (maximum depth), and the refresh buffer is initialized to the background intensity. Each surface listed in the polygon tables is then processed, one line at a time, calculating the depth, or z value, at each (x, y) position. The calculated z value is compared to the value previously stored in the depth buffer at that position. If the calculated z value is compared to the value previously stored in the depth buffer, the new z value is stored, and the surface intensity at that position is placed in the same location in the refresh buffer.

We can summarize the steps of a depth-buffer algorithm as follows:

1. Initialize the depth buffer and refresh buffer so that for all coordinate positions

   depth(x,y) = 1

and

   refresh(x,y) = background.

2. For each position on each surface, compare depth values to previously stored values in the depth buffer to determine visibility.

a. Calculate the z value for each (x,y) position on the surface.
b. If $z < depth_{(x,y)}$, then set $depth_{(x,y)} = z$ and $refresh_{(x,y)} = i$,
where i is the value of the intensity on the surface at position (x,y).

In the last step, if z is not less than the value of the depth buffer for the position, the point is not visible. When the process has been completed for all surfaces, the depth buffer contains z values for the visible surfaces and the refresh buffer contains only the visible intensity values.

Depth values for a position (x,y) are calculated from the plane equation for each surface:

$$z = (-Ax - By - D) / C$$

For any scan line, x coordinates across the line differ by 1 and y values between lines differ by 1. If the depth of position (x, y) has been determined to be z, then the depth z' of the next position (x+1, y) along the scan line is obtained as

$$z' = (-A(x+1) - By - D) / C$$

or

$$z' = z - (A / C)$$

The ratio A/C is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single subtraction.

Depth values between scan line are obtained in a similar manner. Again, suppose that position (x, y) has depth z. Then at position (x, y-1) on the scan line immediately below, the depth value is calculated from the plane equation as

$$z'' = (-Ax - B(y-1) - D) / C$$

or

$$z'' = z + (B / C)$$

which requires a single addition of the constant (B/C) to the previous depth value z.

The depth-buffer method is easy to implement, and it requires no sorting the surfaces in a scene. But it does require the availability of a second buffer in addition to the refresh buffer. A system with a resolution of 1024 by 1024, for example, would require over a million positions in the depth buffer, with each position containing enough bits to represent the number of z -coordinate increments needed.

2.6.3 DEPTH SORTING METHOD

It is possible to use both image-space and object-space operations in a hidden-surface algorithm. The depth-sorting method is a combination of these two approaches that performs the following basic functions:

1. Surfaces are sorted in order of decreasing depth.
2. Surfaces are scan-converted in order, starting with the surface of greatest depth.

Sorting operations are carried out in object space, and the scan conversion of the polygon surfaces is performed in image space.

This method for solving the hidden-surface problem, we first sort surfaces according to their distance from the view plane. The intensity values for the farthest surface are then entered into the refresh buffer. Taking each succeeding surface in turn (in decreasing-depth order), we "paint" the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.

Painting polygon surfaces onto the frame buffer according to depth is carried out in several steps. On the first pass, surfaces are ordered according to the largest z value on each surface. The surface with the greatest depth (call it S) is then compared to the other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S is can converted. This process is then repeated for the next surface in the list. As long as no overlaps occurs, each surface is processed in depth order until all have been scan-converted. If a depth overlap is detected at any point in the list, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.

For each surface that overlaps with S, we make the following tests. If any one of these tests is true, no reordering is necessary for the surface. The tests are listed in order of increasing difficulty.

1. The bounding rectangles in the x-y plane for the two surfaces do not overlap.
2. Surface S is on the "outside" of the overlapping surface relative to the views plane.
3. The overlapping surface is on the "inside" of surface S, relative to the view plane.
4. The projections of the two surfaces onto the view plane do not overlap.

As soon as one test is found to be true for an overlapping surface, we know that surface is not behind S. So we proceed to the next surface that overlaps S. If all the overlapping surfaces pass at least one of these tests, no reordering is necessary and S can be scan-converted.

Test 1 is performed in two parts. We first check for overlap in the x direction, then for overlap in the y direction. If either of these directions shows no overlap, the two planes cannot obscure each other.

We can perform test 2 by substituting the coordinates for all vertices of S into the plane equation for the overlapping surface and testing the sign of the result. Suppose that the overlapping surface has plane coefficients A', B', C' and D' .
If $A'x + B'y + C'z + D' > 0$ for each vertex of S, then surface S is "outside" the overlapping surface. As noted earlier, the coefficients A', B', C' and D' must be specified so that the normal to the overlapping surface points away from the view plane.
Inside test 3 is carried out using the plane coefficients A', B', C' and D' for surface S. If the coordinates for all vertices of the overlapping surface satisfy the condition $Ax + By + Cz + D < 0$, then the overlapping surface is "inside" surface S (providing the surface normal to S points away from the view plane).

If tests 1 through 3 have all failed, we try test 4 by checking for intersections between the bounding edges of the two surfaces using line equations in the xy plane. Note that, two surfaces may or may not intersect even though their bounding volumes overlap in the x, y, and z directions.

Should all four tests fail with a particular overlapping surface S', we interchange surfaces S and S' in the sorted list. However, we still do not know for certain that we have found the farthest surface from the view plane. The figure given below illustrates a situation in which we would first interchange S and S". But since S" obscures part of S', we need to interchange S" and S' to get the three surfaces into the correct depth order. Therefore, we need to repeat the testing process for each surface that is reordered in the list.

It is possible for the algorithm just outlined to get into an infinite loop if two or more surfaces alternately obscure each other, as in the following figure. In such situations, the algorithm would continually reshuffle the positions of the overlapping surfaces. To avoid such loops, we can flag any surface that has been reordered to a farther depth position so that it cannot be moved again. If an attempt is made to switch the surface a second time, we divide it into two parts at the intersection line of the two planes that are under comparison. The original surface is then replaced by the two new surfaces, and we continue processing as before.

2.7 SHADING TECHNIQUES USED

2.7.1 DIRECT SHADING

This technique is used to place shading on surfaces without detailed recourse to correct mathematical calculations. The shading is done by calculating the normal to the plane of the surface which will be shaded. The surfaces then is identified according the direction of the normals to the planes and were linearly interpolated within the given range of colour values.

2.7.2 GOURAUD SHADING

This intensity interpolation scheme, developed by Gouraud, removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane so that intensity values match at the plane boundaries. In this method, intensity values along each scan line passing through a

surface are interpolated from the intensities at the intersection points with the surface.

Intensity values at the vertices of each polygon are determined from a shading model. All other intensities for the surface are then calculated from these values. As illustrated in the figure given below, the intensity at the left intersection of the scan line with the polygon (point 4) is interpolated from the intensities at vertices 1 and 2 with the calculation

$$I_4 = I_1 (y_4 - y_2) / (y_1 - y_2) + I_2 (y_1 - y_4) / (y_1 - y_2)$$

For this example, the intensity at point 4 has a value that is closer to the intensity of point 1 than that of point 2. Similarly, intensity at the right intersection of the scan line (point 6) is interpolated from intensity values at vertices 2 and 3. Once these bounding intensities are established for a scan line, an interior point (such as point 5) is interpolated from the bounding intensities at points 4 and 6 with the calculation.

$$I_5 = I_4 (x_6 - x_5) / (x_6 - x_4) + I_6 (x_5 - x_4) / (x_6 - x_4)$$

This process is repeated for each scan line passing through the polygon.

With this interpolation method, surface normals must first be approximated at each vertex of a polygon. This is accomplished by averaging the surface normals for each polygon containing the vertex point. These vertex normal vectors are then used in the shading model to generate the vertex intensity values. When color is used to shade surface, the intensity of each color component is calculated at the vertices. The method can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.

Gouraud shading removes the intensity discontinuities associated with the constant shading model, but it has some other deficiencies. Highlights on the surface are sometimes displayed with anomalous shapes, and the linear intensity interpolation can cause bright or dark intensity streaks, call Mach bands, to appear on the surface. These effects can be reduced by dividing the surface into a greater number of

polygon faces or by using other methods, such as Phong shading, that require more calculations.

LIST OF MODULES AND FUNCTIONS

ANIMATE.C

---

void doauxiliary(HANDLE hWnd)

Update auxiliary dialog box;

---

long FAR PASCAL auxDlgProc()

Auxiliary dialog box procedure;

---

long FAR PASCAL auxilliaryproc()

Auxiliary window procedure;

BOOL freadmy(FILE *fp, LPSTR lpointer, long count)

Read "count" bytes from a file to a buffer point by lpointer;
Count could be greater than 64K;
Return 1 if OK, otherwise 0;

BOOL fwritemy(FILE *fp, LPSTR lp, long count)

Write "count" bytes to a file from a buffer point by lpointer;
Count could be greater than 64K;
Return 1 if OK, otherwise 0;

DWORD PASCAL lread (HANDLE fh, VOID FAR *pv, DWORD ul)

As freadmy, instead of file pointer is file descriptor;

DWORD PASCAL lwrite (HANDLE fh,VOID FAR *pv, DWORD ul)

As fwritemy, instead of file pointer is file descriptor;

void animateLogAddFrame(DISPLIST *dispList)

Add one more frame to log structure;

void animateLogSetDispList(DISPLIST *dispList, int frame)

Set back position of recorded frame to display list;

HANDLE animateLogStart(DISPLIST *dispList)

Starting animation;
Return handle to log structure;

int animateLogEnd(DISPLIST *dispList)

Finishing animation;
Storing the log file and deleting memory for log structure;

void animateprocess(DISPLIST *dispList)

Update animate dialog box;

LPSTR anigetfilename(int which)

Get a file name, in which the image is stored;

void animateWriteBits(HANDLE fp, LPSTR ptr, long size)

Write image to a file;

void animateReadBits(HANDLE fp, LPSTR ptr, long bitsize)

Read image from a file;
fp file handle;
ptr pointer to bits;
bitsize number of bits;

HANDLE ReadDibBitmapInfo (HANDLE fh)

Read the bitmap informations;
fh is handle to bitmap structure;

BOOL DibInfo (HANDLE hbi,LPBITMAPINFOHEADER lpbi)

Get some informations from device independent bitmap;

HANDLE OpenDIB (LPSTR szFile)

Open bitmap independent file;
Return a file handle;

HANDLE DibFromBitmap (HBITMAP hbm)

Convert a bitmap to a device independent bitmap;

BOOL WriteDIBits (LPSTR szFile, HANDLE hdib)

Write a DIB to a file;
szFile file name;
hdib bitmap handle;

void savebits(HBITMAP bm, int which)

Save a bitmap to a file;

void animatePaintBuffer(DISPLIST *dispList, ZDisplay *disp, int which)

Slow mode painting for animation process;
dispList display list;
disp Z buffer;
which the frame to be painted;

void aniScreenToFile(DISPLIST *dispList)

Add whatever on screen to a log file;

void animatePlayBack(DISPLIST *dispList)

Play back process, after rendering geometry positions;

BOOL FAR PASCAL aniOptions()

Animate option dialog box procedure;

int aniupdatepicture(HANDLE hwnd, int which)

Update the image on screen;
hwnd the screen window;
which the frame umber to be updated;

BOOL FAR PASCAL Animate()

Animate dialog box procedure;

long FAR PASCAL animateproc()

Animate window procedure;

void doanimate(HANDLE hwnd)

Animate call function;
hwnd parent window handle;

void animaterepeat()

Automatic process function;

BEAPINIT.C

BOOL initWindowClass(hInstance)

Init class for every window used in viewbeap;

---

int registerUserWind()

Register for every window used in viewbeap;

BEAPMAIN.C

---

long FAR PASCAL BeapWndProc( )

Window procedure for the main window;

---

void destroyAll()

Clear memory and destroy windows before the application quits;

---

void mainGenImage()

Main general image function;
dispList pointer to display list;

---

long FAR PASCAL AxisWndProc()

Axis window procedure;

BPCOLOR.C

---

short setupcolor()

Creating 128 colours palette for drawing routine;

BPFILE.C

---

void mainOverlay(DISPLIST *displList,HDC image)

Over lay process function to display legends;

---

long FAR PASCAL CommandHandle()

Process every messages pass by WM_COMMAND;

BOOL FAR PASCAL About()

About dialog box procedure;

HANDLE FAR PASCAL LoadDlg()

Open dialog box procedure;

HANDLE FAR PASCAL SaveDlg()

Save dialog box procedure;

BPRESULT.C

ResultPlane *resultGetPlane(int which)

Return pointer to result plane structure if it exists;

---

int resultEnumPlanes()

Enumerates each result plane;
Call back funtion:
(*func)(short , result plane pointer, char *data);

---

void resultDelPlane(int which)

Delete a plane;
"which" ID of the plane to be deleted;

---

int resultAddPlane(ResultPlane *plane)

Add a new result plane to the list;

---

char *resultFName()

Get the result plane file name;

---

void resultLoadPlanes()

Read result plane from a file;

---

int resultSavePlanes()

Save result planes to a file;

COPY.C

BOOL MEM_CMP(char *m1, char *m2, long size)

Compare to block of memory;
Return 1 if they are identical, otherwise 0;

---

void MEM_CPY(char *dst,char *src,unsigned int size)

Copy size bytes from src to dest;

---

void FAR *MEM_ALLOC(long size)

Locate a size bytes from global heap;

---

void FAR * MEM_FREE(void FAR *pointer)

Free the memory block point by pointer;

---

void FAR *MEM_REALLOC(void FAR *pointer, long size)

Realloc a memory block point by pointer to the new size;

BOOL bitmaptoclip(HWND hWnd, int X,int Y,int X2, int Y2)

Copy the image on screen to clip board;
X, Y upper left corner of the image;
X2, Y2 lower right corner of the image;
hWnd handle of the window, which has the image;

BOOL cliptoscreen(HWND hWnd,int X,int Y, int X2, int Y2)

Invert process: copy from clip board to a screen;

DATA.C

ELEMCODE *dataGetEcode(unsigned int n)

Return the pointer to element n code;

EXTRADATA *dataGetExtra()

return pointer to extraData, which presents the coordinate offsets;

NODEDATA *dataGetNode(unsigned int n)

Get pointer to the node number n;

ELEMDATA *dataGetElem(unsigned int n)

Get pointer to the element number n;

---

unsigned int dataGetNumNodes()

return total number of nodes;

---

unsigned int dataGetNumElems()

return total number of elements;

---

void dataDelExtra()

Delete extra data;

void dataClear()

Clear memory for nodes and elements;

---

void dataAddExtra(float xOff, float yOff, float zOff)

Set the offsets for geometry cordinates;

---

unsigned int dataAddNode(unsigned int n, float x, float y, float z)

Add a new node to the node list;

---

unsigned int dataAddElem(unsigned int n, short sides, unsigned int *nodes, short color, char *description)

Add a new element to the element list;

---

void dataCalcNodeVisibility(DISPLIST *dispList)

Set the visibility for every node;

DIALOGMY.C

BOOL GetDlgItemFloat(HWND hDlg,short IDMnumber,float *number)

Get a float number from dialog box control;

---

void SetDlgItemFloat(HWND hDlg, short IDMnumber, float number)

Set a float number to dialog box control;

BOOL GetDlgItemInteger(HWND hDlg,short IDMnumber,int *number)

Get a short number from dialog box control;

DISP.C

---

void dispRecalc()

Set the recalculate function to be actived;

---

void dispRepaint()

The image need to be repainted;

---

void dispListMoved()

The animate process is activated;

---

long dispListSetOverlay()

Set the over lay process function;
DISPLIST *dispList;
void (*overlayProc) ();

---

long dispListSetMoveProc()

Set the animate process function;
DISPLIST *dispList;
void (*moveProc) ();

---

void dispListSetAngles(DISPLIST *dispList, float angXY, float angYZ)

Set the angles XY and YZ to display list;

---

void dispListGetAngles(DISPLIST *dispList, float *angXY, float *angYZ)

Get the angles from display list;

---

void dispListSetOrigin(DISPLIST *dispList, float x, float y, float z)

Set the origin center of the image;

---

void dispListSetViewSize(DISPLIST *dispList, long viewSize)

Set new view size in display list;

---

void dispListSetShading(dispList, shadedModel)

Set shading model for the image;

HANDLE dispListCreate(HWND parent,HWND view, void (*recalcProc)())

Create new display list;
Return handle to the created display list;

HANDLE dispListDestroy(HANDLE hdispList)

Destroy a existing display list;

---

void dispDraw(DISPLIST *dispList, HDC image, int width, int height)

Fast mode drawing routine;

---

void dispDrawBuffered(DISPLIST *dispList,HDC image, int width, int height)

Slow mode drawing routine;

void dispPrintBuffered (DISPLIST *displList,HDC image, int width, int height, int scale)

Slow mode printing routine;

---

int dispCheckResultPlanes(int id, ResultPlane *plane, struct extremesTag *extremes)

Check for the coordinate of plane, correct the extrems if necessary.

---

void dispGenImage()

Generate image:
calculate extrems, center;
generate objects;

---

void dispGenAxisOverlay(DISPLIST *displList,HDC image)

Generate over lay process for axis window;

---

void dispGenAxis(DISPLIST *displList)

Generate the axis image;

---

void dispPaint(DISPLIST *displList, HDC image, int width, int height)

Paint the image on screen;

---

void dispPrint(DISPLIST *displList, HDC image, int width, int height, int scale)

Send the image to the printer;

void dispForceRecalc(DISPLIST *dispList)

Recalculate process procedure;

void dispUpdate(DISPLIST *dispList)

Update the image;

DRAW.C

---

void xv_text(HDC hdc,int x, int y, int color, int op,HFONT font,LPSTR str)

Draw a text point by str;
hdc device context;
x, y screen coordinate;
color text color;
font selected font;
the back ground mode is transparent;

---

void xv_polygon_2(HDC hdc,int *numPts,pr_pos *vlist,int color,int op)

Draw a polygon and fill the interior with color;
numPts number of sides;
vlist coordinate array ;

---

void xv_vector(HDC hDC,int x1,int y1,int x2,int y2,int op,int pixel)

Draw a vector from point (x1, y1) to point (x2, y2);

ELEMST.C

---

float elemStressGetStress(int which,StressTypes type, int x, int y)

return stress value;

---

void elemstfree()

Free the memory allocated for stresses;

BOOL elemStressLoad(int miningStep)

Read stresses , locations and displacements from elesol files;

---

void domeshsize(HWND hwnd,DISPLIST *dispList)

Call the mesh size dialog box;
hwnd parent window handle;
displist current display list;

---

long FAR PASCAL meshsizeproc()

Mesh size dialog box procedure;

---

void elemDispGenMeshCoeff(float (*coeff)[9], int ix, int iy, float f)

Generate the mesh size coefficences;
ix, iy mesh size;
f a factor depends on element libCode (2/3 or .8);

---

void elemDispMult(float *U, float N[9], float (*Ui)[9], int n)

Calculate the real node coordinates;

---

void elemDispGenElemMesh(int elemNum, float (*nodes)[4], int calcStress, float factor, int calcDisp, int stressType, float (*coeff)[9], int ix, int iy)

Generate new elements inresult of mesh size process;

elemNum th base element number;
(*nodes)(4) new coordinates;
calcStress if regards to stresses;
calcDisp if regards to displacements;

EYEWND.C

BOOL initeye()

Init the icon menu for set eye window;

---

void updateeyepalet()

Update icon menu;

---

void updateeyeaction()

Pull down the icon selected;

---

void draweye(HANDLE hwnd,int x, int y)

Draw the eye;
x, y position of the eye;

---

void drawex(HANDLE hwnd,int x, int y)

Draw the eye destination;

---

void reseteye()

Reset parameters for display list;

---

void eyeset(DISPLIST *dispList,BOOL flag)

Since only one display list is used for all windows,
it is needed to set parameter for each window;

---

void eyesetang(float x, float y)

Save the old angles xy and yz;

---

void eyeresetang()

Set back the angles to display list;

---

void correctrect(HANDLE hwnd)

Limit the selected rect inside window;

---

void moveeyeorex(HANDLE hwnd,BOOL flag)

Move eye location or destination;

---

void FAR PASCAL getresultrect()

Eye command event procedure;

---

void doeyeok(DISPLIST *dispList)

Set parameters to the main display list when user click on OK button;

---

void eyecommandhandle()

Processes all WM_COMMAND messages;

long FAR PASCAL EyeWndProc()

Window procedure for main eye window;

long FAR PASCAL EyePaletWndProc()

Window procedure to manipulate icon menu;

long FAR PASCAL EyeXYWndProc()

Window procedure for XY window;

long FAR PASCAL EyeXZWndProc()

Window procedure for XZ window;

long FAR PASCAL EyeYZWndProc()

Window procedure for YZ window;

long FAR PASCAL EyeISOWndProc()

Window procedure for ISO window;

GETPOINT.C

---

void drawkrzyz(HANDLE hwnd,int x, int y)

Draw the indicator by position x and y in one window;

---

void drawkrzyzall()

Draw the indicator in all windows;

---

void initgetpoint(HANDLE hdlg,float x,float y,float z)

Set the position of the indicator to dialog box control;

---

int getPointState()

Handle the messages when the indicator is moving;

---

int getPointStress(float fx, float fy, float fz)

Call m507w modul to calculate stresses by given position ;

---

void getPointCalcStress(HANDLE hdlg)

Calculate stresses inside the cube and by calling m507w;

---

void getPointSetLists(HANDLE hdlg,float *value, int count)

Set the stress values in dialog box;

BOOL FAR PASCAL getpointcubedlg()

Dialog box procedure for updating result cube stresses;

void getpointsetcube(HANDLE hdlg)

Update the cube dialog box;

BOOL FAR PASCAL getpointm507dlg()

Dialog box procedure for updating result m507 stresses;

void getpointsetm507(HANDLE hdlg)

Update the m507 dialog box;

BOOL FAR PASCAL getpointdlg()

Get point dialog box procedure ;

void getpoint(HANDLE hwnd)

Update the get point dialog box;

GETST.C

---

static int _divideGetRealCoors()

Get real coordinates of given point;
RegionInfo *infoInformation about the region;
float *fx, *fy, *fz The real coordinates;
int x, y, z The cell coordinates;

---

static int _findCell(FindValInfo *info)

Find cell and stress values at this cell;

---

static int _getCell(FindValInfo *info, int which)

Get stress values from calculated cube;

---

float getStressValue(FindValInfo *rootInfo, float fx, float fy, float fz, float *distance)

Get stress value by given point (fx, fy, fz);

---

char *getStressFName(int which, int miningStep)

Get result cube file name;

FindValInfo *getStressLoadCube(FindValInfo *info, int regionNumber)

Load result cube from rescube file to info structure;

FindValInfo *getStressLocateCube(FindValInfo *rootInfo, float fx, float fy, float fz)

Get stress by given point (fx, fy, fz) if any;

FindValInfo *getStressStart(StressTypes type, int miningStep)

Starting finding stress process;
Allocate memory for info structure;

---

void getStressEnd(FindValInfo *info)

Free memory resident by info structure;

CalcCube *getStressCalcCube(int which)

Get a cube information;

---

int getStressNumCalcCubes()

return number of cubes;

BOOL getStressEnumCalcCubes(int (*func) (), char *data)

Enumerates all result cubes. A callback function is called for
every cube and must be declared as follows:

CallBack(int id, CalcCube *cube, char *data);

---

void getStressDelCalcCube(int which)

Deletes a cube with ID which;

---

int getStressAddCalcCube()

Adds a new cube and return an ID for this new cube;
CalcCube *cube cube to be added;

---

void getStressUpdateCalcCube(int which, CalcCube *data)

Replace old cube by a new one;

---

void getStressLoadCalcCubes()

Read calculate cubes from a file;

---

void getStressSaveCalcCubes()

Save all cubes to a file;

---

void findStressMaxMin()

Compares stress extrems with cube stress extrems;

GRAPHWND.C

void FAR PASCAL dogeometria()

Window procedure to precess message for zooming, paning and rotation;
DISPLIST *dispList the main display list;
DISPLIST *dispList2 the axis display list;

long FAR PASCAL paintgraphWndProc()

Paint window procedure;

void drawbox(HWND hWnd,BPRECT rect,int x,int y,BOOL draw)

Draws a red box for selecting a region on a screen;
hwnd window handle;
rect the last rectangle selected;
x, y the current mouse position;

void drawline(HWND hWnd,BPRECT rect,int x,int y,BOOL draw)

Draws a red line indicating distance to be paned;
hwnd window handle;
rect the last rectangle selected;
x, y the current mouse position;

void dozooming(HWND hWnd,DISPLIST *dispList,InputBox result)

Performs zooming in or out;
hwnd window handle;
dispList display list to be zoomed;
result region to be zoomed;

void dorotating(HWND hWnd,DISPLIST *displList,BPRECT rect)

Performs rotating;
hwnd window handle;
displList display list to be rotated;
rect contains information about the angles;

void dopaning(HWND hWnd,DISPLIST *displList, BPRECT rect)

Performs paning;
hwnd window handle;
displList display list to be paned;
rect contains information about the distance;

void dokeything()

Interactive key board;
WORD wParam key borad command;
DISPLIST *displList the main display list;
DISPLIST *axisdispList the axis display list;

HIDEPL.C

BOOL inithideplane()

Inits memory for icon memu;

---

void updatehideplanepalet()

Updates the icon menu;

---

void updatehideplaneaction()

Pulls down the selected icon menu;

BOOL selectResultSetVisibility(DISPLIST *displList, DISPPLATE *plate)

Set the visibility of the plate of the plane;

void selectResultShowHideState(HANDLE hwnd, DISPLIST *displList, InputBox results)

Get the selected plate by selecting a rectangle;
results selected rectangle;

void FAR PASCAL dosethideplane()

Processes mouse message to select a region on a screen;

---

int selectResultDrawPlane()

Draw a plane;
int id ID of the plane to be drawed;
ResultPlane *plane plane information
DISPLIST *displList current display list;

void hideplanewndcommand()

Processes all commands in hide plane window;

long FAR PASCAL HidePlaneWndProc()

Hide plane window procedure;

long FAR PASCAL hideplanepWndProc()

Hide icon menu window procedure;

long FAR PASCAL painthideplaneWndProc()

Hide paint window procedure;

long FAR PASCAL axishideplaneWndProc()

Hide axis window procedure;

HIDEWND.C

BOOL inithide()

Inits memory for the icon menu;

---

void updatehidepalet()

Updates the icon menu;

---

void updatehideaction()

Updates the selected icon;

---

void hidePlateSetColor(DISPLIST *displList, DISPPLATE *plate)

Sets the green color for unhide plate and
gray color for the hide plate;

---

void hidePlateState(HANDLE hwnd, DISPLIST *displList, InputBox results)

Selects a region to be hide or unhide;

---

void hidePlateUpdateData(DISPLIST *displList, DISPPLATE *plate, char *data)

Updates data according to the color of the plate;

---

void FAR PASCAL dosethide()

Processes mouse movement messages for set hide window;

void hidewndcommand()

Performs all commands in set hide window;

long FAR PASCAL HideWndProc()

Set hide window procedure;

long FAR PASCAL hidepWndProc()

Icon menu window procedure;

long FAR PASCAL painthideWndProc()

Paint window procedure;

long FAR PASCAL axishideWndProc()

Axis window procedure;

LEGEND.C

---

void legendChangeType(HWND hDlg, WORD wParam)

Changes how to calculate stress color;
wParam IDM_AUTOMATIC
IDM_SEMI
IDM_MANUAL

---

void legendchangefont(HWND hDlg, WORD wParam)

Changes what font will be used to update legend text;
wParam IDM_SMALLFONT
IDM_MEDIUMFONT
IDM_LARGEFONT

---

void legendchangelocation(HWND hDlg, WORD wParam)

Changes where the stress legend will be updated;
wParam IDM_UPPERLEFT
IDM_UPPERRIGHT
IDM_LOWERLEFT
IDM_LOWERRIGHT

---

void legendchangetextlocation(HWND hDlg, WORD wParam)

Changes where the text legend will be updated;
wParam IDM_TUPPERLEFT
IDM_TUPPERRIGHT
IDM_TLOWERLEFT
IDM_TLOWERRIGHT

BOOL legendokey(HANDLE hdlg)

Saves information when user clicks on OK button;

void legendGetOld(HANDLE hwnd)

Gets the old information;

short legendCheckChange(HANDLE hwnd)

Checks wether some thing has been changed in legend dialog box;

BOOL FAR PASCAL legendDlgProc()

Legend dialog box procedure;

short dolegend(HWND hwnd)

Updates legend dialog box;
If successfull return 1;
Otherwise return 0;

HFONT legendGetFont(HDC image)

Gets the seleted font to update legend text;

void legendTextOverlay(DISPLIST *dispList, HDC image)

Overlay procedure to update text legend ;

void legendOverlay(DISPLIST *dispList,HDC image)

Overlay procedure to update stress legend ;

int legendGetStressColor(float value, StressTypes type)

Gets stress color for given value;

OPENFILE.C

---

int loadFile()

Read geom1.dat and geom2.dat files
Those two file contain geometry informations;

---

void FAR PASCAL dispError(HWND hwnd, LPSTR buf)

Display message box with error message in there;

---

int saveFile()

Write to geom1.dat and geom2.dat files
Those two file contain geometry informations;

---

int saveAll(char *dir)

Save all viewbeap informations to files;
char *dir directory in which files will be stored;

---

int loadAll(dir)

Read all viewbeap data file indirectory dir;

PAINTWND.C

BOOL initpaint()

Inits memory for icon menu;

---

void updatepalet()

Updates icon menu;

---

void updateaction(int id)

Pulls down selected icon;

BOOL setColorOfPlate(DISPLIST *displList, DISPPLATE *plate)

Sets the selected color to given plate;

---

void setpaintcolor(HANDLE hwnd,DISPLIST *displList,InputBox results,int colornr)

Selects region to change color;

---

void setColorUpdateData(DISPLIST *displList, DISPPLATE *plate, char *data)

Updates the change;

void dosetcolor()

Processes mouse movement messages to set the region to be selected;

void paintwndcommand()

Performs all command inpaint window;

long FAR PASCAL PaintWndProc()

Set color window procedure;

---

long FAR PASCAL paintp1WndProc()

Icon menu window procedure;

---

long FAR PASCAL paintp2WndProc()

Color window procedure;

---

long FAR PASCAL axispaintWndProc()

Axis window procedure;

---

long FAR PASCAL paintpaintWndProc()

Paint window procedure;

PALET.C

---

void getAction(int x, int y, PALLET NEAR *pallet)

Gets action in icon menu;
x, y mouse coordinate;
pallet information about icon menu;

---

void palletup(PALLET NEAR *pallet,int *x,int *y)

Gets the position of the previouse selected icon;

---

void palletdn(PALLET NEAR *pallet,int *x,int *y)

Gets the position of the current selected icon;

PALLET NEAR *paletinit(HWND hwnd,int offx, int offy, int width, int height,int wn, int hn, int oldaction, int action)

Inits icon menu;
hwnd window contains the menu;
offsetx offset from left in window;
offsety offset from top in window;
width width of a single icon;
height height of a single icon;
wn icon number in x direction;
hn icon number in y direction;

---

void paletupdate(PALLET NEAR *palet, HWND hwnd, short *tableup)

Updates icon menu;
All button will be up;

void paletupdateaction(HWND hwnd, PALLET NEAR *palet, short *Tableup, short *Tabledn)

Updates the selected menu item;

BOOL paletmouse()

Takes care of messages for icon menu;

PERSWND.C

BOOL initpers()

Inits icon menu for perspective mode;

---

void updateperspalet()

Updates icon menu;

---

void updatepersaction(BOOL flag)

Updates the selected icon;

---

long FAR PASCAL PersWndProc()

Perspective window procedure;

PLANEWND.C

BOOL initplane()

Inits icon menu for plane window;

---

void updateplanepalet()

Updates icon menu;

---

void updateplaneaction()

Updates the selected item;

---

int checkplane(HANDLE hwnd)

Check which window is active;

XY window return 1;
XZ window return 2;
YZ window return 3;
ISO return 0;

---

void defResultUpdateChildren()

Updates all windows;

BOOL defResultDelPlane(DISPLIST *dispList, DISPPLATE *plate)

Deletes selected plane;
plate of the plane to be deleted;

BOOL defResultDelState()

Processes messages to select and delete a plane;

BOOL defResultMoveStartResult(DISPLIST *displList, DISPPLATE *plate)

Drags the selected plane;
plate of the plane to be moved;

void _defResultOverPaintPlane(HANDLE view, ResultPlane *plane)

Paints a plane in one window;

void defResultOverPaintPlane(ResultPlane *plane)

Paints a plane in all windows;

int defResultMoveState()

Processes messages to select and move a plane;

void doresultsize()

Performs necessary calculation for sizeing a plane;

int defResultSizeState()

Processes messages to size a plane;

void defResultSetPlaneLoc()

Sets the new location of plane;

void defResultSetPlaneSize()

Sets a new size of the plane;

void defResultSetPanelData(HANDLE hdlg,ResultPlane *plane)

Sets characteristics of a plane to a dialog box;

BOOL planegetvalue(HANDLE hdlg,ResultPlane *plane)

Gets values from dialog box to result plane;

BOOL FAR PASCAL editresultDlg()

Result plane dialog box procedure;

void defResultEditPlaneData(int planeId)

Edit data of a plane with ID planeId;

BOOL defResultEditPlane(DISPLIST *dispList,DISPPLATE *plate)

Selects plane, which has this plate selected;

void defResultAddPlane()

Adds a new result plane;

void defResultDrawPlane(int id,ResultPlane *plane, DISPLIST *dispList)

Draws a plane , which has an ID id;

---

void defResultDrawCube(int id, CalcCube *cube, DISPLIST *dispList)

Draws a cube, which has an ID id in plane window;

---

void defResultGenImage(DISPLIST *dispList)

Generates image in plane windows;

---

void resetplane()

Resets display list parameters to th begining state;

---

void planesetpananglesview(HANDLE hwnd,DISPLIST *dispList)

Sets the pan and view size of display list, depends
which window will be updated;

---

void planesavepanview(HANDLE hwnd,DISPLIST *dispList)

Save the pan and view size ;

---

int defResultEditState()

Takes care of messages in plane window;

BOOL FAR PASCAL maxminDlgProc()

Show max min stress value dialog box procedure;

---

int doshowmaxmin(HANDLE hWnd,DISPLIST *dispList)

Shows the max and min stress values;

void planecommandhandle()

Processes all command in plane window;

long FAR PASCAL PlaneWndProc()

Window procedure for plane main window;

long FAR PASCAL PlanePaletWndProc()

Window procedure for icon menu;

void doplanegeo(HWND hwnd,unsigned message,WORD wParam,LONG lParam)

Performs geometry movements;

long FAR PASCAL PlanXYWndProc()

Window procedure for XY window;

long FAR PASCAL PlanXZWndProc()

Window procedure for XZ window;

long FAR PASCAL PlanYZWndProc()

Window procedure for YZ window;

long FAR PASCAL PlanISOWndProc()

Window procedure for ISO window;

PRINT.C

---

void printInitFirst()

Initial for the first time when the aplication starts running;
Read from win.ini all installed printers;

---

void printChangePrinter(HANDLE hwnd)

Updates the printer setup dialog box;

BOOL FAR PASCAL changePrintDlg()

Dialog box procedure for set up printer dialog box;

HDC getPrinterDC()

Return the handle to the selected printer;

BOOL FAR PASCAL GetInitializationData(HWND hwnd, short which)

Load the driver library, setup printers;

BOOL FAR PASCAL AbortDlg()

Abort dialog box procedure , uses to abort printing process;

---

int FAR PASCAL AbortProc(HDC hPr, int Code)

Takes care of massages during printing;

---

void fontCorrect(HDC hdc)

Sets default printer font during printing;

---

void fontSetBack()

Sets back the selected font before printing;

---

short printOptions(DISPLIST *dispList)

Updates the print dialog box;

BOOL FAR PASCAL PrintDlgProc (hwnd, message, wParam, lParam)

Dialog box function for print dialog box;

---

void doprint(DISPLIST *dispList)

Performs printing process;

RESWND.C

BOOL initresult()

Inits icon menu for result window;

---

void updateresultpalet()

Updates icon menu;

---

void updateresultaction()

Updates selected icon;

---

void defRegionUpdateChildren()

Updates all windows;

BOOL defRegionDelRegion(DISPLIST *dispList, DISPPLATE *plate)

Deletes selected cube;
plate of the cube to be deleted;

BOOL defRegionDelState()

Processes messages to select and delete a cube;

BOOL defRegionMoveStartRegion(DISPLIST *dispList, DISPPLATE *plate)

Drags the selected cube;
plate of the cube to be moved;

---

void _defRegionOverPaintCube(HANDLE hwnd, CalcCube *cube)

Paints a cube in one window;

---

void defRegionOverPaintCube(CalcCube *cube)

Paints a cube in all windows;

---

void defRegionMoveState()

Processes messages to select and move a cube;

---

void doregionsize(HANDLE hwnd, DISPLIST *dispList, int *flag)

Performs necessary calculation for sizeing a cube;

---

int defRegionSizeState()

Processes messages to size a cube;

---

void cubeeditinit(HWND hdlg,CalcCube *cube)

Sets cube values to cube dialog box;

BOOL cubegetvalue(HWND hdlg,CalcCube *cube)

Gets values from dialog box to result cube;

BOOL FAR PASCAL editregionDlg()

Result cube dialog box procedure;

---

void defRegionEditRegionData(int cubeId)

Edit data of a cube with ID cubeId;

BOOL defRegionEditRegion(DISPLIST *displList,DISPPLATE *plate)

Selects a cube, which has this plate selected;

---

void defRegionAddRegion()

Adds a new result cube;

BOOL defRegionDrawCube(int id, CalcCube *cube, DISPLIST *displList)

Draws a cube , which has an ID id;

---

void defRegionGenImage(DISPLIST *displList)

Generates image in result cube windows;

---

void resetcube()

Resets display list parameters to th begining state;

---

void cubesetpananglesview(HANDLE hwnd,DISPLIST *displList)

Sets the pan and view size of display list, depends
which window will be updated;

---

void cubesavepanview(HANDLE hwnd,DISPLIST *displist)

Save the pan and view size ;

---

int defregioneditstate()

Takes care of messages in result cube window;

---

void cubecommandhandle()

Processes all command in result cube window;

---

long FAR PASCAL ResultWndProc()

Procedure for result cube main window;

---

long FAR PASCAL ResultPaletWndProc()

Procedure for result cube icon menu window;

---

long FAR PASCAL ReXYWndProc()

Procedure for result cube XY window;

---

long FAR PASCAL ReXZWndProc()

Procedure for result cube XZ window;

---

long FAR PASCAL ReYZWndProc()

Procedure for result cube YZ window;

long FAR PASCAL RelSOWndProc()

Procedure for result cube ISO window;

SELECT.C

---

int selectClearObj(DISPLIST *dispList, DISPGENOBJ *obj, char *data)

Clear the select bit for a object;

---

void selectClearAll(DISPLIST *dispList, int type)

Clear the select bit for every object;

BOOL selectIsInRect(BPRECT *rect, int x, int y)

Checks if point(x, y) is inside given rectangle;

---

int selectRectCheckObj(DISPLIST *dispList, DISPGENOBJ *obj, BPRECT *rect)

Checks if an object is inside given rectangle;

---

void selectInRect(DISPLIST *dispList, int type, BPRECT *rect)

Checks if an object is inside given rectangle;

---

int selectPointCheckObj(DISPLIST *dispList, DISPGENOBJ *obj, SelectPoint *pt)

Checks if the select point is inside given object;

---

void selectAtPoint(DISPLIST *dispList, int type, int x, int y)

Checks if the select point is inside an object;

---

int selectEnumFunct(DISPLIST *displList, DISPGENOBJ *obj, SelectEnumData *data)

Enumerate function to check if a point is inside an object;

---

void selectEnum(DISPLIST *displList,DispPrimitiveType type,void (*funct) ())

Enumerates every object and find objects inside a region;

---

void selectEnumFirst(DISPLIST *displList,DispPrimitiveType type,void (*funct) ())

Enumerates every object and find the first object, which contains select point;

SOLID.C

BOOL initsolid()

Inits icon menu;

void updatesolidpalet(HANDLE hwnd)

Updates icon menu;

---

void updatesolidaction(HANDLE hwnd)

Updates the selected icon;

---

long FAR PASCAL SolidDlgProc()

Dialog box procedure for solid dialog box;

---

long FAR PASCAL solidwndproc()

Window procedure for window , which has icon menu;

SPEED.C

BOOL initspeed()

Inits icon menu;

---

void updatespeedpalet(HANDLE hwnd)

Updates icon menu;

---

void updatespeedaction(HANDLE hwnd)

Updates the selected icon;

---

long FAR PASCAL speedDlgProc()

Dialog box procedure for speed dialog box;

---

long FAR PASCAL speedproc()

Window procedure for window , which has icon menu;

STEP.C

BOOL initstep()

Inits icon menu;

---

void updatesteppalet(HANDLE hwnd)

Updates icon menu;

---

void updatestepaction(HANDLE hwnd)

Updates the selected icon;

---

long FAR PASCAL StepDlgProc()

Dialog box procedure for step dialog box;

---

long FAR PASCAL stepwndproc()

Window procedure for window , which has icon menu;

STRESST.C

BOOL initstresstype()

Inits icon menu;

---

void updatestresstypepalet(HANDLE hwnd)

Updates icon menu;

---

void updatestresstypeaction(HANDLE hwnd)

Updates the selected icon;

---

void setlaststresstype(DISPLIST *dispList)

Sets the stress type selected;

---

int getlaststresstype()

Gets the stress type selected;

---

long FAR PASCAL StresstypeDlgProc()

Dialog box procedure for stress type dialog box;

---

long FAR PASCAL stresstypeproc()

Window procedure for window , which has icon menu;

STRESSV.C

BOOL initstressview()

Inits icon menu;

---

void updatestressviewpalet(HANDLE hwnd)

Updates icon menu;

---

void updatestressviewaction(HANDLE hwnd)

Updates the selected icon;

---

long FAR PASCAL StressviewDlgProc()

Dialog box procedure for stress view type dialog box;

---

long FAR PASCAL stressviewproc()

Window procedure for window , which has icon menu;

THREED.C

BOOL FAR PASCAL threeDDlg()

Dialog box procedure for threeD dialog box;

---

int dodistance(HANDLE hwnd, DISPLIST *displist, DISPLIST *axisdisplist)

Sets distance between eyes by updating dialog box;